On Sample Selection for Continual Learning: a Video Streaming Case Study

Anonymous authors
Paper under double-blind review

Abstract

Machine learning (ML) is a powerful tool to model the complexity of communication networks. As networks evolve, we cannot rely on a train-once-and-deploy strategy. Retraining models regularly—known as continual learning—is necessary. Networks often generate massive amounts of data, too much to train on regularly. Yet, to date, there is no established methodology to answer the following key questions:
With which samples to retrain? *When should we retrain?*

We address these questions with Memento, a sample selection system for networking ML models. Memento maintains a training set with the "most useful" samples to maximize sample space coverage. This approach benefits rare patterns—the notoriously long "tail" in networking—without hurting the average. Moreover, it allows assessing rationally *when* it may be helpful to retrain, i.e., when the space coverage changes.

We deployed Memento on Puffer, the live-TV streaming project, and achieve a 14 % reduction of stall time, 3.5× the improvement of random sample selection, without significantly impacting video quality. While this paper focuses on Puffer as a case study, Memento's design does not depend on a specific model architecture; it is likely to yield benefits in other ML-based networking applications.

1 Introduction

In video streaming, Adaptive Bit Rate (ABR) algorithms aim to avoid video stalls while maximizing image quality. This entails modeling the network dynamics, a complex task for which researchers have started to use machine learning (ML) [37, 60, 61]. As the field progresses, achieving good average performance is feasible: Current ML-based ABR algorithms achieve high average image quality while largely avoiding stalls [33, 61]. Stalls are the rare distribution tail.

Deploying ML-based ABR faces two important challenges: **maintaining performance over time**–known as continual learning—and **providing good tail performance**. Continual learning [10, 26, 42, 50] is fundamental as networks change (e.g., new protocols get deployed) and new traffic patterns arise (e.g., streaming video over satellite networks). The first key question is not if, but *when should we retrain the models?*

The Puffer project [1] perfectly illustrates these two challenges. To date, it is one of the best longitudinal studies of ML in networking, monitoring ABR performance for real users streaming live TV with randomly-assigned algorithms. Puffer's authors proposed their own ML-based ABR and retrained it daily using random samples from the past two weeks. To the author's surprise [61], retraining every day brought essentially no benefits. Over almost 900 days, a daily-retrained model improved image quality by 0.17 % over a static—never retrained—one (Fig. 1). On the tail, daily retraining reduced the fraction of stream time spent stalled by 4.17 %. In short, daily retraining mostly wasted resources. *But why?*

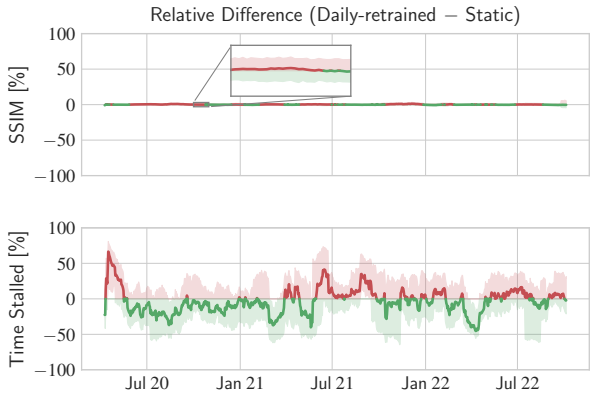


Figure 1: On Puffer [1], retraining daily with a random memory does not consistently outperform a model never retrained. While daily differences are noisy, image quality and stream-time spent stalled differ by less than 0.2 % and 4.2 % overall.
Mean and 90% CI over a one-month sliding window. Data published by [1].

Significantly improving tail performance is challenging due to the heavy-tailed nature of network data: by definition, there are few tail samples, and they are tricky to identify as such. Yet, they matter. For ABR, underestimating path delay may lead to stalls, which are known to impact user experience far more than image quality [15, 31]. To improve a model's tail performance, we must address this training dataset imbalance. Fig. 1 suggests that retraining *can* improve tail performance—since one model *is* performing better—but selecting a random set of relatively recent samples is not reliable. Some days the selection got lucky, and retraining improved the tail; some days, it got much worse. Thus, the second key question is, *with which samples should we retrain?*

In summary, to improve tail performance,i.e., reduce video stalls, we should retrain with "right samples," and only once we gathered sufficiently many to warrant benefits.

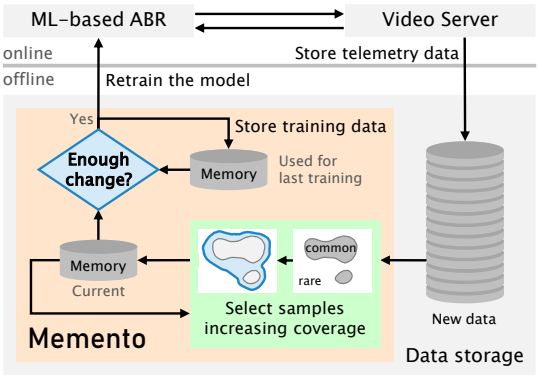


Figure 2: We present Memento, a sample selection system based on *coverage maximization*, i.e., selecting training samples that maximize the sample space's coverage. It improves tail performance while rationalizing when to retrain: one should only retrain when there is enough change in coverage.

Existing approaches Arzani et al. [6] also suggest using the "right samples" to retrain AutoML systems [22]. They use the disagreement among a set of models to identify "the tail" and guide the user to collect more tail samples and add them to the training set. However, this does not apply to ABR. We cannot "enforce" streaming sessions to come from rare network paths or experience particular congestion patterns over the Internet: we must do with the samples available.

One could get more tail samples by simply training on more samples. But this requires more resources and may bring little to no benefits. As we show (Appendix A, Fig. 19), naively using more samples can fail to improve performance because it does not address the imbalance of the training set.

One strategy to address this imbalance is to optimize the training algorithm. One example is Just Train Twice (JTT) [35], which trains a model, uses it to identify misclassified samples, upsamples those, and trains again. This approach performs well, but provides a different trade-off than Memento: it improves tail performance through *additional* training efforts and does not help to decide when to retrain.

Conversely, DriftSurf [53] is designed to detect when to retrain and switches to an entirely new training set when it does. This approach is too coarse-grained for networking: Network traffic is composed of many different patterns (Appendix A, Fig. 9), and not all patterns get outdated at once.

For networking, Matchmaker [36] proposes a more incremental adaptation by maintaining multiple models in parallel and "matching" each sample to the best model to make a prediction for that sample. Fundamentally, this does not address the sample selection problem: If all models perform poorly at the tail, combining them will not help. Alike JTT, it also does not help to decide when to retrain the models. Our evaluation shows that a single model with better sample selection outperforms Matchmaker at the tail, even with a perfect oracle selecting the optimal model for each sample §4.4.

The Problem In this work, we study these two critical questions for the deployment of ML-based ABR solutions:

1. From a continuous flow of new samples, *with which samples should we retrain* to reduce video stalls while maintaining high video quality?
2. From an updated set of training samples in memory, *when should we retrain the model?* Given the large resource costs, can we avoid retraining "for nothing"?

Our solution We address both questions with Memento, a sample selection system based on *coverage maximization*, i.e., prioritizing samples from low-density areas of the sample space. Fig. 2 illustrates how Memento integrates with an ML-based video streaming system, which typically transmits videos in chunks. In ML-based ABR, a *sample* encompasses telemetry data for a video chunk combined with a trace of telemetry data for previous chunks. The video server collects samples as it streams videos to clients. Memento estimates the sample-space density from both new and in-memory samples. Based on this density, prioritizes rare low-density samples over common ones: this addresses dataset imbalance and improves tail performance. Moreover, Memento uses the *difference in coverage* to assess whether the memory has qualitatively changed, i.e., whether it contains samples from which there is something new to learn. If so, it triggers retraining.

Main contributions

- We propose a density-based sample selection system for continual learning called Memento (Fig. 2). We implement Memento and make it publicly available [5].
- We deployed Memento on Puffer: Over more than 9 months and 10 stream-years of real-world data, Memento achieves a 14 % smaller fraction of stream-time spent stalled than the static model; 3.5× the improvement of Puffer's daily retraining with random samples in the past. Image quality is only degraded by 0.13 % and we retrain just 3 times after the first 8 days (7 times in total, §4).
- In additional simulations, we demonstrate that Memento:
  - improves the tail with a small average impact;
  - has easy-to-tune parameters;
  - works with classification and regression problems;
  - and picks up new patterns quickly.

The best of both worlds We discussed that better training methods (e.g., JTT [35]) or model architectures (e.g., Matchmaker [36]) are not sufficient to address all continual learning challenges in ML-based ABR. Still, those ideas are useful and complementary to our work on sample selection.

We show in §4.4 that combining Memento with JTT or Matchmaker improves performance further: once Memento decides to retrain, we train better with JTT, and MatchMaker benefits from an ensemble of Memento-trained models. However, optimizing training or model architectures is beyond the scope of this work, which focuses on *identifying the most valuable samples* for retraining and deciding *when to retrain.*

2 A case for density

This section provides intuition why using density as a selection metric makes sense. We then detail how Memento uses density to maximize sample-space coverage (§3) and demonstrate its effectiveness on Puffer as a case study (§4).

Density for sample selection "The tail" is not a single pattern that rare traffic follows, but rather many patterns with only a few samples each. A random sample selection mirrors any imbalance in the underlying distribution, e.g., over-represented common traffic patterns. This leads to *diminishing returns* as we sample more and more from common patterns and little from the tail. As illustrated by Fig. 1, this yields good average performance but is unreliable at the tail. We need to address the imbalance of the training set.

To correct dataset imbalance, we need a sample-aware selection. Traditional approaches use the model performance (e.g., loss or reward) to select samples [10, 26]. We found that this does not work well on Puffer because it fails to *differentiate rare patterns from noise*.[1] It focuses too much on the high-loss patterns and forgoes the rest of the traffic.

Instead, we propose to select samples based on the density of their neighborhood, which considers the whole sample space. That is, a *sample-space-aware selection* that aims to *maximize coverage* of the sample space. The key insight to maximize coverage is to retain samples where few similar samples exist. Instead, we remove samples from high-density areas with the most similar samples. This decreases the density and we naturally stop removing further samples.

Fig. 3 illustrates the benefits of samples-space-aware density versus sample-aware loss as a selection metric. In this experiment, we use the static model from Puffer's authors and 5 M samples collected by Puffer over a few days. We create batches of 256 samples and compute their loss and density (as detailed in §3). The top row of Fig. 3 shows the mean loss and density per batch. We then use each metric to select 1 M samples and retrain the model. The bottom row compares the effect of each selection metric over the 5 M samples by showing the mean prediction improvement per batch.

The left column shows that loss-based selection focuses *too much* on the tail, i.e., high-loss batches. It yields good improvements there but does not preserve enough average samples and degrades performance on most other batches.

Conversely, the right column illustrates that density-based selection is more holistic, focusing on the tail while covering the entire sample space. Performance is improved at the tail, i.e., on low-density batches, without much degradation on the high-density ones. One may object that the loss selection could be tuned to maintain more "low-loss samples." We tried this and our evaluation shows that it does not perform as well as density and can easily get much worse (§4.4 and Fig. 6).

[1] Consider this analogy of computer vision: the image of an oddly-parked car is rare but learnable, whereas an utterly blurry picture is just noise. Yet both are at "the tail" and have a high model loss.

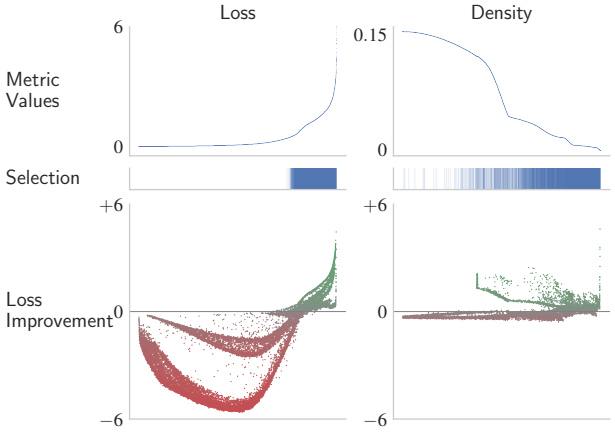


Figure 3: *Loss improvement obtained by retaining with 1M samples over a dataset of 5M. The same batches of 256 samples are used for the loss-based (left) and density-based selection (right).*

To improve tail performance, we need many low-density batches because they are all different. To maintain average performance, we need only a few high-density batches, as they are similar. Selecting based on density (right) achieves both. Conversely, loss-based selection (left) is too specific. It suffers from diminishing returns by selecting too many high-loss batches and catastrophically forgets the average.

Density for shift detection Continual learning aims to adapt when the data distribution changes. These changes can be broadly categorized into *covariate shift* [52], i.e., previously unseen traffic patterns emerge or the prevalence of patterns changes and *concept drift* [57], i.e., the underlying network dynamics change. Density-based sample selection is an effective tool to capture both kinds of changes.

When new traffic patterns appear, e.g., users starting to stream over satellite networks, they populate a previously-empty area of the sample space, resulting in a low density and, thus, a high probability of being selected for retraining. Similarly, patterns becoming less prevalent results in low density, and a high probability of remaining selected.

When the underlying network dynamics change, e.g., a new congestion control gets deployed, we should forget old samples that are no longer relevant. However, detecting those changes is difficult, and being wrong risks forgetting useful information such as hard-to-gather tail samples.

Using density for sample selection *correlates the probability of forgetting samples with how important it is to remember them.* Samples from dense regions are discarded readily, as we will likely get more of those samples. Conversely, low-density batches are less likely discarded as we only encounter these samples infrequently. This makes the selection more conservative at the tail, allowing to remember tail patterns; the model will perform well on similar traffic if it recurs. If it does not, i.e., it was essentially noise, then it will eventually be forgotten. We provide some empirical evidence of recurring patterns in the tail of the Puffer traffic in Appendix A.

3 Coverage maximization

The core of Memento is Algorithm 1: the sample selection algorithm designed to maximize the sample-space coverage. Memento achieves this by estimating the sample space density from the distances between samples: the higher the density of a point in sample space (i.e., the more samples are close to it) the better the memory covers this part of sample space. Memento approximates optimal coverage by iteratively discarding samples, assigning a higher discard probability to high-density regions. It proceeds in five steps:

1. It considers input and output spaces separately, and addresses high-dimensional inputs by comparing them in the low-dimensional *prediction space* (§3.2);
2. For scalability, it computes pairwise distances between *distributions of sample batches* (§3.3);
3. It estimates input- and output-space density using *kernel density estimation* (KDE, §3.4).
4. It discards batches *probabilistically* until fitting the memory constraints. It maps density to discard probability, balancing tail-focus and noise rejection (§3.5).
5. Once new samples are selected, it approximates how much the memory coverage has increased to decide whether retraining might be beneficial (§3.6).

Memento's sample selection relies on three internal parameters: the batch size b, the KDE bandwidth h, and the probability mapping temperature T. We provide default choices for these parameters in §4.2 and analyze the impact of each parameter on Memento's performance in §4.4.

Algorithm 1: Memento Coverage Maximization

Parameters: Capacity C, threshold τ, batch size b, bandwidth h, temperature T
Input: In-memory S_{mem} and incoming S_{new} samples
Output: Selected samples S^*, decision *retrain*

begin
- $S^* \leftarrow S_{new} \cup S_{mem}$
- $B' \leftarrow \{\}$; // or last train batches
- *retrain* $\leftarrow$ *False*
- // §3.2: Dimensionality reduction
- $S_{BBDR} \leftarrow BBDR(S^*)$; // $(x,y) \rightarrow (\hat{y},y)$
- // §3.3: Distance measurement
- $B \leftarrow BatchSamples(S_{BBRD}, b)$
- $(D^{pred}, D^{out}) \leftarrow DistributionDistances(B)$
- // §3.4: Density estimation
- $\hat{\rho}^k \leftarrow KDE(D^k, h)$ $\forall k \in \{pred, out\}$
- $\hat{\rho} \leftarrow \min(\hat{\rho}^{pred}, \hat{\rho}^{out})$
- // §3.5: Sample selection
- **while** $|S^*| > C$ **do**
  - $p^{discard} \leftarrow \text{softmax}(\hat{\rho} / T)$
  - $i \leftarrow WeightedRandomChoice(p^{discard})$
  - $S^*, B \leftarrow DiscardBatch(S^*, B, i)$
  - $\hat{\rho}, D \leftarrow UpdateDensities(\hat{\rho}, D, i)$
- // §3.6: Retraining decision
- **if** $RCI(B, B') \geq \tau$ **then**
  - *retrain* $\leftarrow$ *True*
  - $B' \leftarrow B$; // remember train batches

3.1 Definitions

Process We consider a process $y = f(x)$ that maps inputs $x \in \mathcal{X} = \mathbb{R}^n$ to outputs $y \in \mathcal{Y}$, where $\mathcal{Y} = \mathbb{R}$ in regression or $\{1, 2, \ldots, k\}$ in classification problems. In the context of ABR, f models the network dynamics mapping traffic features x (e.g., video chunk size, TCP statistics, transmission times of past packets) to a prediction for the next chunk y (e.g., the current bandwidth or expected chunk transmission time).

Predictions We consider a model $\hat{f}$ that is trained to predict $\hat{y} = \hat{f}(x)$ from a set of training samples S, where $S = \{(x_1, y_1), \ldots, (x_N, y_N))\}$, i.e., a supervised setting.

Replay memory To account for concept drift or covariate shift, we retrain $\hat{f}$ with an updated set of samples S^*, stored in a *replay memory* with capacity C. A sample selection strategy decides how to update this memory.

Sample selection strategy Given a set S_{new} of new samples available and a set S_{mem} of samples currently stored in memory, with $|S_{new}| + |S_{mem}| > C$, a selection strategy must select $S^* \subset (S_{new} \cup S_{mem})$, such that $|S^*| \leq C$.

3.2 Dimensionality reduction

It is not clear a priori whether coverage in the sample input or output space is most important for tail performance. Thus, in supervised classification or regression problems—where outputs are available—Memento considers both equally. First, it computes distances and densities for input and output space separately; then, it combines these densities to prioritize samples that maximize coverage across both spaces (§3.4).

While outputs are typically one- or low-dimensional, inputs tend to be high-dimensional; e.g., the ML model in Puffer [61] predicts the transmission times of video chunks and has over 60 input dimensions. In high dimensions, distance computation becomes difficult [3, 67]. Manually weighing inputs is unreliable, as it is unclear which dimensions matter most for the model. We address this problem using *black-box dimensionality reduction* (BBDR), an approach inspired by research on dataset shift detection [34] that has been found to outperform alternatives like PCA [46]. The idea is simple: Use the existing model—trained to identify important feature combinations—to predict $\hat{y}$ (the predicted transmission time), then compute distances in the low-dimensional prediction space. In essence, BBDR leverages prediction as dimensionality reduction tailored to the task at hand.

3.3 Distance measurement

Batching Puffer [1] currently collects over 1 M samples daily. A simple approach distance measure, such as computing pairwise euclidean distances, does not scale to such a large dataset. Memento addresses this scalability issue using batching. Instead of computing distances between individual samples, we pool them into batches and compute distances between the empirical *distributions of sample batches*.[2] With a batch size n, we divide the distance computations by $O(n^2)$.

While batching improves scalability, it reduces the flexibility of sample selection. For example, if a common and a rare sample are pooled in the same batch, they can only be kept or discarded together. To avoid suboptimal pooling, samples are first grouped by outputs, then by predictions, and finally split into batches of specified size b; this pools samples spatially to create homogeneous batches. We found this batching to work best, but Memento also supports batching based on sampling time or application-specific criteria.

Distribution distances There are many ways to compute distances between distributions, each giving larger weights to certain types of differences (e.g., in location or spread). In our case, to capture tail performance, we consider the difference in *information*, which gives large weight to rare samples—those with low probabilities in the distributions.

A commonly used measure of information difference is the *Kullback-Leibler Divergence* [32] (KL). However, it is ill-suited to estimate density as it is not a distance metric, is asymmetrical, and can grow infinitely large. Instead, we use the closely related Jensen-Shannon Distance (JSD), which is a symmetrical and bounded distance metric [18].

Definition 1 (Jensen-Shannon Distance)**.** *Let P and Q be probability distributions, and let $M = \frac{1}{2}(P+Q)$. The Jensen-Shannon Distance between P and Q is defined by:*

$$JSD(P,Q) = \sqrt{\tfrac{1}{2}\left(KL(P,M) + KL(Q,M)\right)}$$

$$\textit{where} \qquad KL(P,M) = \sum_{x \in \mathcal{X}} P(x) \log_2 \left(\frac{P(x)}{M(x)}\right)$$

if P and Q are discrete distributions in the space $\mathcal{X}$; or

$$KL(P,M) = \int_{-\infty}^{\infty} p(x) \log_2 \left(\frac{p(x)}{m(x)}\right) dx$$

if P and Q are continuous probability distributions with probability density functions p and q, and $m = \frac{1}{2}(p+q)$.

Probabilistic predictions During batching, Memento leverages probabilistic predictions; e.g., the transition time predictor in Puffer [61] outputs a probability distribution over 21 transition time bins. This probability distribution captures whether predictions are somewhat uncertain—thus indicating a need for training using more such samples—or certain. However, we must handle probabilistic estimates differently than point estimates: *(i)* When batching samples, Memento groups probabilistic predictions first by the distribution mode, then by their probability; *(ii)* The batch distribution is computed as a mixture distribution, i.e., the prediction distribution for batch b_i is $P_i(x) = (1/|b_i|) \cdot \sum P_j(x)$, where P_j is the probabilistic prediction for sample j.

[2] This may also be interpreted as replacing distances between observations (samples) with distances between the underlying processes (distributions).

3.4 Density estimation

Now that we have sample distances, we can proceed to compute the prediction- and output-space densities. Since we do not know the topology of the sample space, we cannot use simple density approximations, such as the fraction of samples per cluster. In such cases, a common approach is to use kernel density estimation (KDE) [43].

Definition 2 (KDE)**.** *Let b be a sample batch and B a set of sample batches, and let $d^k(b,b') = JSD(P^k, P^{k'})$ with $k \in \{pred, out\}$ be the prediction or output distance between batches b and b' with distributions $P^k, P^{k'}$. Then, using a Gaussian kernel with* bandwidth *h, the kernel density estimate $\hat{\rho}$ at the location of batch b is defined as:*

$$\widehat{\rho}_B^k(b) = \frac{1}{\sqrt{2\pi}|B|} \sum_{b' \in B} \exp\left(-\frac{d^k(b,b')^2}{2h^2}\right) \tag{1}$$

Intuitively, the kernel density estimate of b is inversely proportional to the distances to other batches, with diminishing weights for more distant ones. The bandwidth h, the "smoothing factor", determines how quickly this drop-off occurs.

Density aggregation Memento considers the prediction and output space as equally important. Thus, we aggregate densities using the minimum: we regard the batch as rare if its density is low in either the prediction or output space:

$$\widehat{\rho}_B(b) = \min\left(\widehat{\rho}_B^{pred}(b), \widehat{\rho}_B^{out}(b)\right) \tag{2}$$

If necessary, this approach can be generalized to fewer or more densities: for unsupervised learning without ground truth, Memento can only consider $\widehat{\rho}^{pred}$. For models with multiple outputs and thus multiple predictions, we can compute the minimum in Eq. (2) across all respective densities.

3.5 Sample selection

Memento optimizes the sample-space coverage by iteratively discarding high-density batches. It computes the densities for all samples in $S^* = S_{new} \cup S_{mem}$, randomly discards batches with a density-dependent probability (Algorithm 1, Lin. 10–14) until $|S^*| \leq C$, and updates densities after each discard.

Intuitively, Memento assigns a high discard probability to batches with high density—batches for which many similar

samples exist—thereby protecting rare low-density samples. However, because noisy samples also seem "rare," we must retain some probability of discarding rare samples. Memento achieves this by mapping densities to probabilities using softmax with temperature scaling [21]:

$$p^{discard} = \text{soft}\max\left(\widehat{\rho}_B / T\right) \tag{3}$$

where $\widehat{\rho}_B$ is a vector of densities for all batches $b \in B$, and $p^{discard}$ is a corresponding vector of discard probabilities.

The temperature T allows balancing tail-focus with noise rejection. A low temperature assigns a higher discard probability to the highest-density batch(es). Conversely, a high temperature assigns more uniform discard probabilities, increasing the probability of discarding low-density batches. At the extreme, the discard probability with $T \to 0$ is a point mass; if Memento is configured with T=0, we thus deterministically discard the highest-density batch. With $T \to \infty$, the probability becomes uniformly random.

3.6 Retraining decision

Intuitively, retraining is beneficial if we collect *new* information, i.e., samples in areas of the sample space that were previously not covered. That is, we should retrain only when the coverage of sample space increases. Memento's density estimation allows estimating this increase in information to guide the retraining decision.

Definition 3 (Coverage increase)**.** *Let B be a set of batches with density estimates $\widehat{\rho}_B$. We can approximate the region of the sample space covered by the samples in B:*

$$Coverage(B) = \sum_{b \in B} \widehat{\rho}_B(b) \tag{4}$$

Let B' be a second set of sample batches. We can approximate the coverage increase, short CI, of B with respect to B', i.e., the region of sample space covered by B but not by B':

$$CI(B, B') = \sum_{b \in B} \min\left(\widehat{\rho}_B(b) - \widehat{\rho}_{B'}(b), 0\right) \tag{5}$$

The relative *coverage increase RCI from B' to B is then*

$$RCI(B, B') = CI(B, B') / Coverage(B) \tag{6}$$

where $RCI(B, B') \in [0, 1]$; 0 means that the same area of sample space is covered, while 1 indicates that B covers an entirely different region of the sample space than B'.

Hence, with B the current memory batches and B' those used for the last model training, $RCI(B, B')$ estimates the coverage increase since the last training. If it exceeds the user-defined threshold τ, Memento triggers retraining. This approach presents a rational trade-off: the larger τ, the longer we wait for changes to accumulate before retraining. With a small τ, the model compensates for changes quicker at the cost of more retraining. Memento's training decision is sample-aware, it gives a rational argument that retraining is likely to be beneficial (even if there is no guarantee).

4 Evaluation: Real-world benefits

We use Puffer [1] to evaluate Memento's benefits in the real world. This experiment aims to show that Memento improves the tail performance of *existing models* reliably without significantly impacting the average. Puffer provides both a public dataset with data collected daily over several years and a publicly available model that we can retrain with Memento and compare against the original. This makes Puffer a perfect case study to investigate the following questions:

Q1 *Does* Memento *improve the tail predictions?* ***Yes***
Over years of live and replay video data, Memento significantly improves the 1st percentile prediction score.

Q2 *Does it improve the application performance?* ***Yes***
On live Puffer, over 10 stream-years of data, Memento achieves a 14 % smaller fraction of stream-time spent stalled with only 0.14 % degradation in image quality.

Q3 *Does* Memento *avoid unnecessary retraining?* ***Yes***
Memento retrains 4 times in the first 8 days, and only 3 times in the following 9 months (7 times in total).

Q4 *Are our improvements replicable?* ***Most likely***
Memento benefits appear replicable over different time periods of Puffer data. Moreover, its design parameters are intuitive and easy to tune.

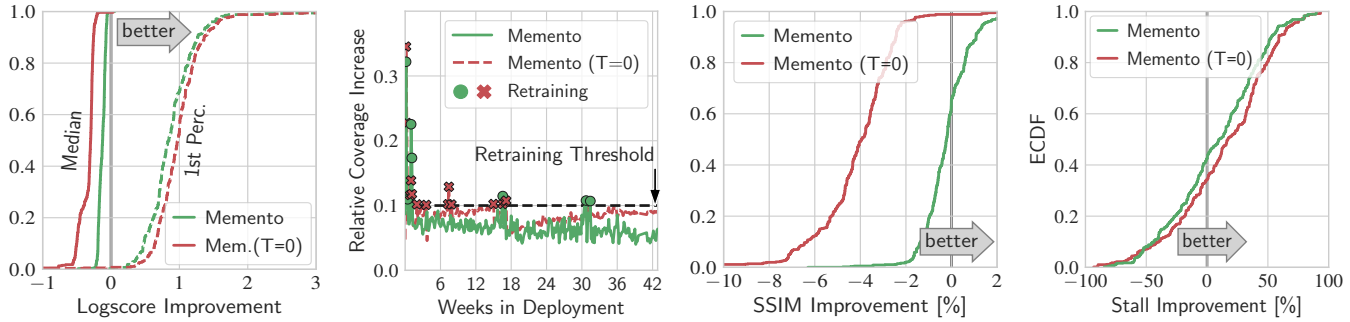
Q5 *Can* Memento *benefit existing solutions?* ***Yes***
Memento further improves tail predictions achieved by more advanced training or prediction strategies.

4.1 The Puffer project

The Puffer project is an ongoing experiment comparing ABR algorithms for video streaming [1]. Puffer streams live TV with a random assignment of ABR algorithms to video sessions and collects Quality-of-Experience (QoE) data for each transmitted chunk of video data: the mean image quality measured in SSIM [66] and the time spent with stalled video.

Fugu is the ABR algorithm proposed by Puffer's authors; it features a classical control loop built around a *Transmission Time Predictor* (TTP), a neural network predicting the probabilities for a set of discretized transmission times. The predictor was retrained every day with 1 M samples drawn randomly from data over the past 2 weeks: ∼130 k "fresh" samples from the last day and ∼10 % fewer samples for each day further into the past. Fugu_{Feb} is a static version of the same algorithm, trained in February 2019 and never retrained.

Fugu was discontinued only 17 days after Memento's current deployment. Hence, we can only compare Memento's long-term performance to Fugu_{Feb}. As discussed in §1, Fugu and Fugu_{Feb} achieve similar performance: Over almost three years, Fugu showed an SSIM improvement of 0.17 % and a reduction in the time spend stalled of 4.17 %. Thus, improvements over Fugu_{Feb} would likely translate to similar—yet slightly lesser—improvements over the daily-retrained Fugu.



(a) Memento improves the prediction quality at the tail, with a relatively small impact on the median.

(b) Memento avoids unnecessary retraining. Its deterministic variant ($T = 0$) is less efficient.

(c) Memento degrades the SSIM slightly (0.14 % worse). The deterministic variant is 4.4 % worse.

(d) Memento reduces the fraction of stream-time spent stalled by 14 %, although results vary by day.

Figure 4: Memento achieves its goal: it improves the tail prediction quality with minimal impact on the average (Fig. 4a). This requires little retraining (Fig. 4b) and translates into modest but notable QoE improvements (Figs. 4c and 4d).

4.2 Retraining with Memento

We use Memento to retrain Fugu's TTP: every day, we use Memento to select the training samples and decide whether to retrain. We assess Memento's benefits in two experiments:

Deployment We deploy on Puffer two Memento variants, i.e., two variants of Fugu using Memento for retraining the TTP. One uses Memento's default parameters (see below), and the other deterministic sample selection (i.e., using temperature $T = 0$, §3.5). We collected data over 292 days (from Oct. 2022 to Jul. 2023), totaling around 10.8 stream-years of video data per variant. This experiment allows answering **Q1**, **Q2**, and **Q3**.

Replay To confirm the deployment observations, evaluate design choices, and benchmark the impact of Memento's parameters, we replay Puffer data collected since 2021. To reduce the bias from a particular starting day, we replay 3 instances with 6 months of video data each and a total of 90 stream-years of video data. This experiment allows answering **Q4** and **Q5**.

Metrics We access Memento along three dimensions:

- *Prediction quality* is measured with the logarithmic score $logscore(y) = \log p(y)$ [20], where y is the transmission time of a video chunk, and $p(y)$ its probability predicted by the TTP;[3] Available in deployment and replay.
- *Application performance* is measured with user QoE; Only available for the deployment experiment, where real user streams are impacted by the predictions and the resulting QoE can be measured.
- *Training resource utilization* is measured by the number of retraining events in deployment and replay.

Parameters We set Memento's memory capacity C to 1 M samples (same as the original Fugu model). Default parameters are a retraining threshold τ of 0.1, batching size b of 256, kernel bandwidth h of 0.1, and temperature T of 0.01. We evaluate the impact of different parameter values in §4.4.

4.3 Deployment results

In this section, we show that Memento effectively improves the tail prediction quality with little retraining and that this translates into QoE improvements over Fugu_{Feb}.

In Appendix A, we provide additional plots with the evolution of QoE and predictions over time, and aggregate plots like those published on the Puffer paper and website [1, 61].[4]

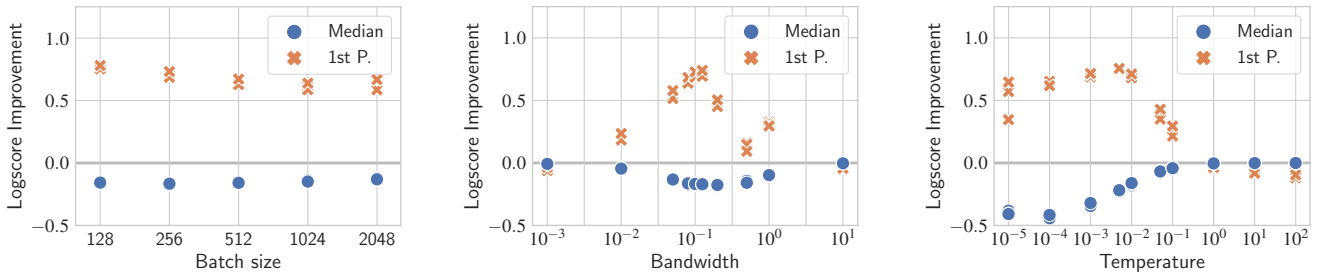
Prediction quality Fig. 4a shows the ECFD of the score difference between Memento and Fugu_{Feb} in median and 1st-percentile scores on the whole deployment (higher is better).

We observe that Memento improves tail predicition performance as intended, with a slight—yet expected—degradation on average: as memory is finite and Memento purposefully prioritizes "rare" samples, it must remove samples for the most common cases. The deterministic version of Memento prioritizes the tail more aggressively, which leads to slightly better tail improvements and worse median degradation.

Retraining count Fig. 4b shows the relative coverage increase *RCI* between the current memory and the one last used to retrain the model, as well as the RCI retraining threshold $\tau = 0.1$. We observe about eight "warm-up" days where Memento retrains four times. Afterward, RCI remains low and retraining due to changes in the data (RCI peaks) happens three times. This shows that there are fewer "new patterns" to learn from over time; retraining daily is unnecessary.

[3] The logarithmic score is a commonly used metric for probabilistic predictions [20] like those produced by the Puffer TTP model. It is closely related to the *cross entropy loss* used to train the TTP: this is also known as the *logarithmic loss* and is the negative logarithmic score.

[4] Our results per algorithm differ from official Puffer plots, as Puffer excludes some sessions in an attempt to exclude effects such as "client decoder too slow," while we consider all data points. As the filtering is ABR-independent, it does not impact relative results between ABRs.



(a) Larger batches slightly reduce tail improvements with almost no impact on the median.

(b) A kernel bandwidth around 0.1 gives the results. Extreme values nullify benefits.

(c) $T = 0.01$ is a good trade-off between tail prioritization, randomness, and robustness.

Figure 5: Each marker shows the median prediction improvement of Memento over Fugu (random selection) over a 6-month replay, measured as median and 1st perc. improvement each day. Results are consistent across replays.

Conversely, the deterministic variant of Memento keeps accumulating samples, exhibiting a different RCI pattern. After training five times during "warm-up," it trained 9 more times. Where RCI for default Memento stabilizes, the RCI of the deterministic variant slowly but steadily increases over time while it accumulates rarer and rarer samples, as it does not reject noise. Essentially, this variant "never forgets."

Application performance Figs. 4c and 4d show the relative QoE improvements achieved over Fugu_{Feb} for the SSIM and the time stalled (higher is better), respectively.[5]

First, we observe that Memento only marginally affects the SSIM; the average SSIM is 17.12 and 17.14 for Memento and Fugu_{Feb}, resp. (not shown). Memento's deterministic variant affects the SSIM more; its average is 16.39 (not shown) and the SSIM is consistently worse than Fugu_{Feb} (Fig. 4c).

Second, Fig. 4d shows that the improvement in stalls compared to Fugu_{Feb} is almost the same for both variants of Memento, even though there are large day-to-day variations: some days, Fugu_{Feb} stalls much less than Memento, and vice versa. Over the entire 292 days of deployment, Memento spent a fraction of 0.2 % of stream-time stalled, compared to 0.24 % for Fugu_{Feb}. In relative terms, Memento spent a 14 % smaller fraction of stream time stalled than Fugu_{Feb}.

In the results above, we already see that Memento performs slightly worse without noise rejection (i.e., with $T = 0$). In a previous deployment, we observed that never forgetting ultimately prevented enough average samples to remain in memory, which destroyed the average performance (Appendix A). The latest version of Memento made the deterministic variant more robust but we see the signs of noise accumulation (more frequent retraining, steadily rising RCI). By contrast, the probabilistic default Memento naturally forgets noise and stabilizes, as can be seen in the RCI in Fig. 4b.

Finally, we observe that Memento reduces stalls by 3.5 times as much as retraining daily with random samples. In Appendix A, Fig. 11 we show Fig. 4a overlaid with the score improvements of Fugu in the past.[6] We observe that random retraining improved the tail prediction scores significantly less and even worsened them on 20 % of days.

From predictions to QoE One may wonder why the average prediction degradation (Fig. 4a) does not seem to strongly impact image quality (Fig. 4c), and, conversely, why significant prediction improvements at the tail yield only a modest reduction in stalls (Fig. 4d). Our results illustrate the complex relationship between prediction quality and QoE, including the closed-loop control logic between the predictions and the chosen chunk size, which aims to keep the video buffer at the receiver sufficiently full to avoid stalling.

Looking closer, we noticed that the transmission time of most chunks is very small, and most prediction errors are also small time-wise. Hence, slightly worse predictions have little effect on the buffer fill level; the controller has time to compensate and maintain image quality. Moreover, since most prediction errors overestimate the transmission time (not shown), it makes the closed-loop control more conservative. Thus, it manages to keep stalls low but struggles to maintain high image quality (compare Figs. 4c and 4d). Further investigations of the interplay between prediction quality and application performance would be interesting but are beyond the scope of this work. To facilitate further research, we publish all our retrained models including their training sample selection alongside the Puffer QoE data.

4.4 Replay results

In this section, we confirm that Memento's benefits are replicable, i.e., they are not just an artifact from deploying at an "easy time," its design decisions are justified, and it complements existing techniques. To do this, we replay 3 non-overlapping instances of 6 months, containing 25, 39 and 26 stream-years of video-data respectively.

[5] To avoid bias towards either Memento or Fugu, we show the symmetric percent difference using the maximum: $100 \cdot (x-y)/\max(x, y)$.

[6] Appendix A, Fig. 11 must be considered with caution, as the underlying data comes from different time periods and may not be comparable.

To monitor the memory quality over time, we disable threshold-based retraining and retrain every 7 days: one must retrain and test the model to assess whether the right samples were selected. We evaluate each day in terms of prediction improvement over retraining with random samples (Fugu) and report the mean over each 6-month period.[7]

Replicability All plots in Fig. 5 show three data points per setting, which are average performance numbers over the entire 6 month period. We observe that all results are fairly stable, which gives reasonable confidence about the replicability of Memento's benefits on this use case.

Batch size Fig. 5a shows Memento's prediction performance over the batch size; larger sizes improve scalability but make sample selection more coarse-grained, which should hurt performance (§3.3). We observe a slight tail performance drop for larger batch sizes with little change on average.

Regarding scalability, differences are more pronounced: using a single CPU core to process 2 M samples[8] takes on average 200 s with a batch size of 128, 48 s with 256 (the default), and 7 s with 1024. Benefits flatten out for larger batch sizes. Overall, computation is dominated by distance computation; batching the samples takes only about 2.5 s.

Bandwidth For each batch, Memento estimates how close nearby batches are; the kernel bandwidth h determines what "nearby" means (§3.4). As the computed distances $JSD(P,Q) \in [0,1]$, bandwidths > 1 over-smooth (all batches are always "nearby"), and bandwidths $\ll 1$ under-smooth (no other batches are ever "nearby"). Both cases nullify the idea of estimating density, effectively making the sample selection random. Fig. 5b confirms this intuition: at the extremes, Memento performs like a random selection. We obtain the best tail improvement with a bandwidth around 1×10^{-1}.

Temperature Fig. 5c shows Memento's prediction performance over the temperature T; a low temperature strongly prioritizes rare samples at the risk of accumulating noise, while a high temperature rejects noise by making the sample selection more random (§3.5). As expected, a lower temperature yields better tail performance but degrades the average. The trade-off is not linear, though; we can select a temperature that provides tail benefits with minimal impact on the average. The best trade-off is a temperature around 1×10^{-2}.

Alternative selection metrics Fig. 6 shows the performance of using loss as an alternative selection metric: we still use temperature-based probabilistic selection but prefer to discard samples with a low loss rather than high density. At best, it gives tail improvements about half as Memento, but it is much harder to tune: the benefits vanish for a slightly higher temperature. With a lower temperature, i.e., selecting more strongly based on loss, the model performance decreases drastically, which mirrors our observations in §2.

[7] We show the entire time series for each experiment in Appendix A.
[8] One million in memory, one million new samples on average per day.

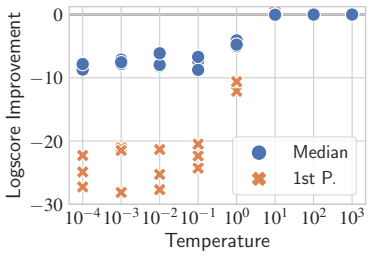


Figure 6: Loss-based selection is worse and less robust.

We evaluate additional metrics in Appendix A: prediction confidence, label counts, and whether a sample belongs to a stalled session or not. In summary, these perform worse or equal to loss-based selection in the best case, and most of them are as sensitive to tune. Probabilistic selection based on density performs better and is less sensitive (see Fig. 5c).

Alternative training decision We also compare Memento's retraining decision based on the *relative coverage increase* RCI with a loss-based decision (not shown). We observe that for samples selected by Memento, either retraining decision is effective. Overall, a coverage-based decision provides greater control over retraining frequency but struggles with low thresholds (e.g., 5 %). As Memento is probabilistic, the estimated RCI fluctuates at each iteration, which can also be observed in Fig. 4b, and the retraining threshold should be set above these fluctuations. It may be possible to further improve Memento's sample selection by smoothing the RCI or by attempting to remove the random fluctuations. We leave this challenge for future work.

Combining Memento with better predictions and training MatchMaker [36] improves predictions by using an ensemble of models (default is 7) combined with an online algorithm to select the best model to make a prediction for each sample. However, this is limited by the performance of the individual models. Using an ensemble of models trained with a random selection, even with an oracle choosing the best model, we can only improve tail performance by half as much as a single model trained with Memento. However, we can get the best of both worlds by using MatchMaker with an ensemble of Memento-trained models, which yields double the tail perfor-

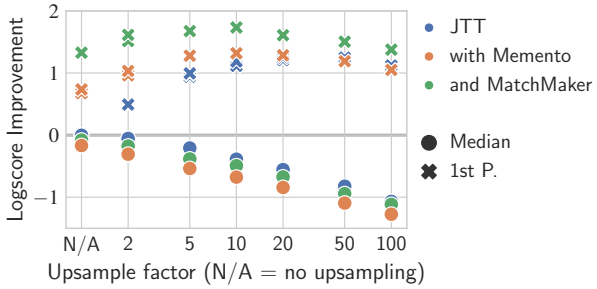


Figure 7: Memento complements training and prediction improvements such as JTT and Matchmaker.

mance with less decrease in median performance compared to a single Memento-trained model ('no upsampling' in Fig. 7).

JTT [35] improves training by training twice: after the first training, misclassified samples are upsampled in the second and final training. Fig. 7 shows the same performance trade-offs for JTT and Memento: it improves the tail and degrades the median: a random selection with a JTT upsampling factor of 3 is roughly equivalent to Memento's sample selection with 'normal' training. Yet this comes at different resource costs: JTT requires up to double the training time and resources, depending on how long the first training step is. Training models like Puffer take time in the range of hours [61] and often require expensive hardware (e.g., GPUs). Memento is more resource efficient: even on a single CPU core, it can process millions of samples in a few minutes (see above). However, JTT and Memento are not in competition, but complementary. We observe the best performance by combining Memento-selected samples with JTT's training, and observe even further improvements when the resulting models are used in a MatchMaker ensemble to make predictions (Fig. 7).

5 Evaluation: Synthetic shifts

In the previous section, we showed that Memento provides significant benefits in a real-world use case, but where network conditions appear relatively stable. Thus, in this section, we use simulation in ns-3 [49] to illustrate Memento's performance on other networking tasks and under substantial distribution shifts. Specifically, we show that Memento:

1. ensures good tail performance by reliably prioritizing samples from infrequent traffic patterns (§5.2);
2. picks up new patterns quickly (§5.3);
3. is applicable to classification and regression (§5.4).

5.1 Experimental setup

Sample selection strategies We compare Memento with two baselines: Random (random sampling) and FIFO (keep recent samples). In addition, we compare it to the state-of-the-art LARS (Loss-Aware Reservoir Sampling, [10]). LARS uses several improvements to random sampling for classification, and has two stages: first, it randomly chooses to keep or discard a new sample, with probability exponentially decreasing over time; second, it considers both label counts and loss to decide which in-memory sample to replace.

Parameters For Memento, we use the same default parameters as before: a batching size of 256, kernel bandwidth h of 0.1, and temperature T of 0.01. We reduce the memory capacity C to 20 k samples (i.e., 1/50 compared to §4) for two reasons: *(i)* We aim to show the limitations of different sample selections strategies, which is easier to do using small memories; *(ii)* LARS scales poorly, making comparing performance on larger memory sizes impractical—we optimized the original LARS implementation to scale to 20 k samples. Our optimization is available in our artifacts.

Workloads The simulation setup (Appendix B, Fig. 12) consists of two nodes and applications sending messages whose sizes follow three empirical traffic distributions from the Homa project (Fig. 13, [39]): Facebook web server (**W1**), DC-TCP (**W2**), and Facebook Hadoop (**W3**). W1 and W3 are similar size-wise, while W2 messages are about an order of magnitude smaller. For each workload, we generate 20 traffic traces of 1 min each. During this time several senders transmit a combined 20 Mbps, resulting in an average network utilization of 66 %. We repeat this process by injecting additional cross traffic to reach an average utilization of 100 % and 133 %, respectively. We use different random initializations to generate a total of 180 distinct runs that we combine in various iterations in the following experiments.

Models We compare the selection strategies for two neural networks; one for classification and one for regression. The classification model predicts the application workload: Each input is a trace of the past 128 application packet sizes, and the model predicts the probabilities for each workload. The regression model predicts the next transmission time from past packets: Each input contains a trace of the past 127 packet sizes and transmission times, the current packet size, and the model predicts the transmission time. See Appendix B for architecture and hyperparameter details.

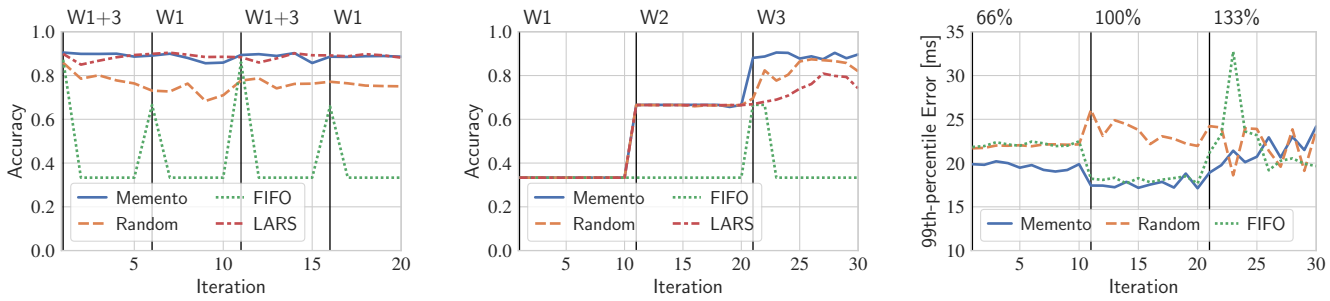
Metrics For classification tasks, we measure the balanced accuracy, i.e., the accuracy obtained over an equal number of evaluation samples per workload, ensuring equal importance of each workload. In other words, performance is evaluated over an equal distribution of overall workloads, regardless of whether they are present at the current iteration. Good performance requires both picking up new patterns quickly and avoiding catastrophic forgetting.

For regression, we investigate changes in traffic distribution (see below) and measure the 99th percentile absolute prediction error over the data in the latest iteration. Good performance requires picking up new patterns quickly.

5.2 Classification: Rare patterns

In the first experiment, we show that Memento successfully picks up samples from infrequent traffic patterns. To do so, we use highly imbalanced traffic: W1 and W3 only constitute $\leq$2 % of overall traffic. Good tail performance implies high accuracy not only for W2 but also for W1 & W3.

Setup We use the classification model and iterate over samples from 20 runs. We use W2 at every iteration, representing a large part of traffic that remains relatively unchanged. On top of that, we include W1 once every five iterations, and W3 once every ten iterations; they represent sporadic traffic patterns that make up for 1.3 and 0.5 % of traffic respectively.



(a) Rare and infrequent traffic patterns: Only LARS and Memento avoid forgetting. W2 is always present; W1 and W3 only where marked.

(b) Incremental learning: Memento efficiently integrates workload traffic patterns as they appear sequentially where marked.

(c) Increasing congestion: Memento has comparable or lower 99th percentile error for the marked network utilization.

Figure 8: Memento can handle various traffic patterns and prediction types, outperforming other approaches.

Results Memento and LARS retain sufficient samples from each workload and show the best accuracy over all iterations (Fig. 8a). On the other hand, FIFO shows good accuracy only while all workloads are present, as the large number of samples of W2 quickly overwrites W1 & W3 otherwise. While Random achieves better results than FIFO, it ultimately retains too few samples of W1 & W3, as they make of less than 2 % of samples in memory.

5.3 Classification: Incremental learning

Next, we show that Memento quickly picks up new patterns and avoids catastrophic forgetting in an 'Incremental Learning' setting, which is known for its challenging nature [19].

Setup We use the same model and setup as §5.2, but iterate over samples from each workload sequentially; first W1, then W2, and finally W3, for 10 iterations each.

Results We find that overall, Memento exhibits the best performance. Both Random and LARS struggle because of their sample selection rate (Fig. 8b); these two flavors of random memory avoid forgetting by decreasing the probability of selecting new samples over time. When W3 is introduced, they are slow to incorporate new samples (Appendix B, **??**).[9] By manually tuning the sampling rate of LARS to be much more aggressive, we were able to achieve the same performance as Memento (not shown). This highlights the benefit of the self-adapting nature of Memento's sample-space-aware approach: If a new label appears, Memento discovers that this part of the sample space is not well covered yet. It quickly prioritizes discarding common in-memory samples to retain samples from the new label.

[9]While both LARS and Random are slow to react, the fact that the balanced accuracy of LARS is worse than Random's is mostly an artifact of the similarity of W1 and W3: LARS has (desirably) retained more samples of W1, yet this causes its model to mistake W3 for W1 more often than Random, which has forgotten most of W1 and is consequently less biased.

5.4 Regression

In this experiment, we show that Memento is applicable to regression and handles complex traffic changes. We iterate from 66 to 133 % network utilization, which presents more complex gradual changes in traffic patterns than the abrupt changes in workload distributions (§5.2 and §5.3).

Setup We iterate over traffic from all workloads using runs with increasing congestion. For the first 10 iterations, we use runs with 66 % network utilization, followed by 10 iterations of 100 %, and finally 10 iterations of 133 %. We report the 99th percentile error under the current traffic conditions.

Results Memento generally shows the lowest 99th percentile prediction error (Fig. 8c). Random is slow to react to the new patterns and requires several iterations to adjust to the new traffic conditions. Perhaps surprisingly, FIFO performs well up to 100 % utilization but shows very unstable performance for 133 %. LARS is not applicable to regression.

6 Related work & Discussion

Generalization Memento could be applied to other ML-based networking applications, including congestion control [2, 27, 41, 58], traffic optimization [11], routing [54], flow size prediction [16, 45], MAC protocol optimization [28, 63], traffic classification [9, 56], network simulation [65], or DDoS detection [56]. Networking has proven to be a challenging environment for ML, and many proposed systems have only delivered modest or inconsistent improvements in real networks [7, 8, 61, 62]. In response, research has focused on providing better model architectures [2, 27, 61] and training algorithms [60], model ensembles for predictions and active learning [22, 36], real-world evaluation platforms [61, 62], uncertainty estimation [51] and model verification [17].

A better sample selection is beneficial to all these advances. Memento is orthogonal to and complements these works, opening an exciting potential. For example, our evaluation showed that combining Memento with MatchMaker outperforms each of the individual solutions (§4.4). We look forward to future research investigating, e.g., how to design ML models to maximize coverage maximization benefits.

Continual learning Fundamentally, continual learning suffers from the *stability-plasticity* dilemma [14]: a stable memory consolidates existing information yet fails to adapt to changes, while a plastic memory readily integrates new information at the cost of forgetting old information. Forgetting old-yet-still-useful information is known as *catastrophic forgetting* [38]. Continual learning approaches aim to be as plastic as possible while minimizing catastrophic forgetting. They can be broadly categorized as either prior-based or rehearsal-based [10]. Prior-based methods aim to prevent catastrophic forgetting by protecting model parameters from later updates [30, 42, 64]. Rehearsal-based methods collect samples over time in a replay memory and aim to prevent forgetting by learning from both new and replayed old data [10, 26, 47]. Hybrid methods combine both, e.g., training with a replay memory and an additional loss term penalizing performance degradation on old samples [50].

Memento builds on previous rehearsal-based approaches, incorporating ideas such as coverage maximization [13]. It extends existing approaches ideas by considering both prediction and output spaces, leveraging temperature scaling to control the tail-focus, as well as introducing a novel *coverage increase* criterion to reason about when to retrain.

Distribution shift detection Continual learning closely relates to a branch of research aiming to keep models up-to-date upon changes in the data-generating process—known as distribution shifts. State-of-the-art methods rely on statistical hypothesis testing [24] or changes in empirical loss [53], plus a time-based window (or multiple parallel windows) [14]. When a change is detected, these algorithms advance the time window(s), discard "outdated" samples whose timestamp falls outside of the window(s), and retrain.

Shift detection algorithms make *sample-aware* retraining decisions yet lack a sample-aware selection strategy. Once a change is detected, they discard all old samples, ill-suited to the diversity of networks, where only a small subset of traffic may change at any given time. Memento's sample selection based on coverage maximization is sample-aware for *both* sample selection and retraining decisions. To do so, it leverages an idea originating from shift detection: BBDR [34].

Experience replay for reinforcement learning While we evaluate Memento in the context of supervised learning, it may also be used for *reinforcement learning* (RL), which is popular approach to ML-based ABR [37, 60]. In fact, RL commonly uses a replay memory [48, 55], and it has been shown that it can greatly improve RL performance [4, 50].

Data processing Data validation & augmentation are important steps of any ML pipeline. Especially in ML systems that are evolving over time, new bugs may be introduced any time the model or data collecting system are updated. These bugs may lead to erroneous data, and data validation is necessary to prevent such data from becoming part of the training data [44]. Furthermore, collected data is often augmented to address dataset imbalance by generating additional synthetic samples or upscaling existing ones (e.g., JTT [35]), which can improve performance and reduce overfitting [59]

As a replay memory, Memento operates between the validation and augmentation steps, and complements them. For example, we showed in §4.4 that JTT's upsampling is more effective when Memento has identified tail samples first. Only validated data should be considered by the sample selection, and selected samples may be augmented. In particular, the memory should only store non-augmented samples, as augmenting the data before passing it to the memory can result in the sample selection to overfit to the augmentation [10].

Limitations Let us first address the elephant in the room: Memento cannot do anything with a bad model or dataset. It helps identify the most useful samples for training, but those samples must be present in the dataset in the first place, and the model must be capable of learning from them.

We find that density-based selection performs well, but it is probably not optimal. It addresses the problem of dataset imbalance well, but it is less effective to differentiate "hard-to-learn" from "easy-to-learn" samples, which is better captured by the model loss. Combining both would be likely beneficial.

Finally, Memento's scalability is limited by its density computations. Batching helps, but would not be enough to process data streams with billions of samples daily.

Ethical issues This work does not raise any ethical issues.

References

[1] Puffer. URL: https://puffer.stanford.edu/.

[2] Soheil Abbasloo, Chen-Yu Yen, and H. Jonathan Chao. Classic Meets Modern: A Pragmatic Learning-Based Congestion Control for the Internet. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 632–647, Virtual Event USA, July 2020. ACM. URL: https://dl.acm.org/doi/10.1145/3387514.3405892, https://doi.org/10.1145/3387514.3405892.

[3] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the Surprising Behavior of Distance Metrics in High Dimensional Space. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory — ICDT 2001*, Lecture Notes in Computer Science, pages 420–434, Berlin, Heidelberg, 2001. Springer. https://doi.org/10.1007/3-540-44503-X_27.

[4] Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. Online Continual Learning with Maximal Interfered Retrieval. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 11849–11860. Curran Associates, Inc., 2019. URL: http://papers.nips.cc/paper/9357-online-continual-learning-with-maximal-interfered-retrieval.pdf.

[5] Anonymous. Reference blinded due to double-blind submission.

[6] Behnaz Arzani, Kevin Hsieh, and Haoxian Chen. Interpretable Feedback for AutoML and a Proposal for Domain-customized AutoML for Networking. In *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*, HotNets '21, pages 53–60, New York, NY, USA, November 2021. Association for Computing Machinery. https://doi.org/10.1145/3484266.3487373.

[7] Eytan Bakshy. Real-world Video Adaptation with Reinforcement Learning. April 2019. URL: https://openreview.net/forum?id=SJlCkwN8iV.

[8] Mihovil Bartulovic, Junchen Jiang, Sivaraman Balakrishnan, Vyas Sekar, and Bruno Sinopoli. Biases in Data-Driven Networking, and What to Do About Them. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, pages 192–198, New York, NY, USA, November 2017. Association for Computing Machinery. https://doi.org/10.1145/3152434.3152448.

[9] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. pForest: In-Network Inference with Random Forests, September 2019. URL: http://arxiv.org/abs/1909.05680, arXiv:1909.05680, https://doi.org/10.48550/arXiv.1909.05680.

[10] Pietro Buzzega, Matteo Boschini, Angelo Porrello, and Simone Calderara. Rethinking Experience Replay: A Bag of Tricks for Continual Learning. *arXiv:2010.05595 [cs, stat]*, October 2020. URL: http://arxiv.org/abs/2010.05595, arXiv:2010.05595.

[11] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. AuTO: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 191–205, New York, NY, USA, August 2018. Association for Computing Machinery. https://doi.org/10.1145/3230543.3230551.

[12] François Chollet et al. Keras. 2015. URL: https://keras.io.

[13] T. de Bruin, J. Kober, K. Tuyls, and R. Babuška. Improved deep reinforcement learning for robotics through distribution-based experience retention. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3947–3952, October 2016. https://doi.org/10.1109/IROS.2016.7759581.

[14] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. Learning in Nonstationary Environments: A Survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, November 2015. https://doi.org/10.1109/MCI.2015.2471196.

[15] Zhengfang Duanmu, Kai Zeng, Kede Ma, Abdul Rehman, and Zhou Wang. A Quality-of-Experience Index for Streaming Video. *IEEE Journal of Selected Topics in Signal Processing*, 11(1):154–166, February 2017. https://doi.org/10.1109/JSTSP.2016.2608329.

[16] Vojislav Dukić, Sangeetha Abdu Jyothi, Bojan Karlas, Muhsen Owaida, Ce Zhang, and Ankit Singla. Is advance knowledge of flow sizes a plausible assumption? In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 565–580, Boston, MA, February 2019. USENIX Association. URL: https://www.usenix.org/conference/nsdi19/presentation/dukic.

[17] Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. Verifying learning-augmented systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, pages 305–318, New York, NY, USA, August 2021. Association for Computing Machinery. https://doi.org/10.1145/3452296.3472936.

[18] D. M. Endres and J. E. Schindelin. A new metric for probability distributions. *IEEE Transactions on Information Theory*, 49(7):1858–1860, July 2003.

[19] Sebastian Farquhar and Yarin Gal. Towards Robust Evaluations of Continual Learning. *arXiv:1805.09733 [cs, stat]*, June 2019. URL: http://arxiv.org/abs/1805.09733, arXiv:1805.09733.

[20] Tilmann Gneiting and Adrian E. Raftery. Strictly Proper Scoring Rules, Prediction, and Estimation. *Journal of the American Statistical Association*, 102(477):359–378, March 2007. https://doi.org/10.1198/016214506000001437.

[21] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On Calibration of Modern Neural Networks, August 2017. URL: http://arxiv.org/abs/1706.04599, arXiv:1706.04599, https://doi.org/10.48550/arXiv.1706.04599.

[22] Xin He, Kaiyong Zhao, and Xiaowen Chu. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, January 2021. URL: https://www.sciencedirect.com/science/article/pii/S0950705120307516, https://doi.org/10.1016/j.knosys.2020.106622.

[23] Tim Head, MechCoder, Gilles Louppe, Iaroslav Shcherbatyi, fcharras, Zé Vinícius, cmmalone, Christopher Schröder, nel215, Nuno Campos, Todd Young, Stefano Cereda, Thomas Fan, rene-rex, Kejia (KJ) Shi, Justus Schwabedal, carlosdanielcsantos, Hvass-Labs, Mikhail Pak, SoManyUsernamesTaken, Fred Callaway, Loïc Estève, Lilian Besson, Mehdi Cherti, Karlson Pfannschmidt, Fabian Linzberger, Christophe Cauet, Anna Gut, Andreas Mueller, and Alexander Fabisch. Scikit-optimize/scikit-optimize: V0.5.2. Zenodo, March 2018. URL: https://zenodo.org/record/1207017, https://doi.org/10.5281/zenodo.1207017.

[24] Fabian Hinder, André Artelt, and Barbara Hammer. Towards Non-Parametric Drift Detection via Dynamic Adapting Window Independence Drift Detection (DAWIDD). In *Proceedings of the 37th International Conference on Machine Learning*, pages 4249–4259. PMLR, November 2020. URL: https://proceedings.mlr.press/v119/hinder20a.html.

[25] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, pages 448–456. PMLR, June 2015. URL: http://proceedings.mlr.press/v37/ioffe15.html.

[26] David Isele and Akansel Cosgun. Selective Experience Replay for Lifelong Learning. *arXiv:1802.10269 [cs]*, February 2018. URL: http://arxiv.org/abs/1802.10269, arXiv:1802.10269.

[27] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A Deep Reinforcement Learning Perspective on Internet Congestion Control. In *Proceedings of the 36th International Conference on Machine Learning*, pages 3050–3059. PMLR, May 2019. URL: https://proceedings.mlr.press/v97/jay19a.html.

[28] Suraj Jog, Zikun Liu, Antonio Franques, Vimuth Fernando, Sergi Abadal, Josep Torrellas, and Haitham Hassanieh. One Protocol to Rule Them All: Wireless {Network-on-Chip} using Deep Reinforcement Learning. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 973–989, 2021. URL: https://www.usenix.org/conference/nsdi21/presentation/jog.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017. URL: http://arxiv.org/abs/1412.6980, arXiv:1412.6980.

[30] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, March 2017. URL: https://www.pnas.org/content/114/13/3521, https://doi.org/10.1073/pnas.1611835114.

[31] S. Shunmuga Krishnan and Ramesh K. Sitaraman. Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-Experimental Designs. *IEEE/ACM Transactions on Networking*, 21(6):2001–2014, December 2013. https://doi.org/10.1109/TNET.2013.2281542.

[32] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951. URL: https://www.jstor.org/stable/2236703.

[33] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 183–196, New York, NY, USA, August 2017. Association for Computing Machinery. https://doi.org/10.1145/3098822.3098842.

[34] Zachary C. Lipton, Yu-Xiang Wang, and Alex Smola. Detecting and Correcting for Label Shift with Black Box Predictors. *arXiv:1802.03916 [cs, stat]*, July 2018. URL: http://arxiv.org/abs/1802.03916, arXiv:1802.03916.

[35] Evan Z. Liu, Behzad Haghgoo, Annie S. Chen, Aditi Raghunathan, Pang Wei Koh, Shiori Sagawa, Percy Liang, and Chelsea Finn. Just Train Twice: Improving Group Robustness without Training Group Information. In *Proceedings of the 38th International Conference on Machine Learning*, pages 6781–6792. PMLR, July 2021. URL: https://proceedings.mlr.press/v139/liu21f.html.

[36] Ankur Mallick, Kevin Hsieh, Behnaz Arzani, and Gauri Joshi. Matchmaker: Data Drift Mitigation in Machine Learning for Large-Scale Systems. *Proceedings of Machine Learning and Systems*, 4:77–94, April 2022. URL: https://proceedings.mlsys.org/paper/2022/hash/1c383cd30b7c298ab50293adfecb7b18-Abstract.html.

[37] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 197–210, Los Angeles, CA, USA, August 2017. Association for Computing Machinery. https://doi.org/10.1145/3098822.3098843.

[38] James L. McClelland, Bruce L. McNaughton, and Randall C. O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102(3):419–457, 1995. https://doi.org/10.1037/0033-295X.102.3.419.

[39] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 221–235, Budapest, Hungary, August 2018. Association for Computing Machinery. https://doi.org/10.1145/3230543.3230564.

[40] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, Haifa, Israel, June 2010. Omnipress. URL: http://www.icml2010.org/papers/432.pdf.

[41] X. Nie, Y. Zhao, Z. Li, G. Chen, K. Sui, J. Zhang, Z. Ye, and D. Pei. Dynamic TCP Initial Windows and Congestion Control Schemes Through Reinforcement Learning. *IEEE Journal on Selected Areas in Communications*, 37(6):1231–1247, June 2019. https://doi.org/10.1109/JSAC.2019.2904350.

[42] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, May 2019. URL: http://www.sciencedirect.com/science/article/pii/S0893608019300231, https://doi.org/10.1016/j.neunet.2019.01.012.

[43] Emanuel Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

[44] Neoklis Polyzotis, Martin Zinkevich, Sudip Roy, Eric Breck, and Steven Whang. Data Validation for Machine Learning. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems*, volume 1, pages 334–347, 2019. URL: https://proceedings.mlsys.org/paper/2019/file/5878a7ab84fb43402106c575658472fa-Paper.pdf.

[45] P. Poupart, Z. Chen, P. Jaini, F. Fung, H. Susanto, Yanhui Geng, Li Chen, K. Chen, and Hao Jin. Online flow size prediction for improved network routing. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pages 1–6, November 2016. https://doi.org/10.1109/ICNP.2016.7785324.

[46] Stephan Rabanser, Stephan Günnemann, and Zachary Lipton. Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 1396–1408. Curran Associates, Inc., 2019. URL: http://papers.nips.cc/paper/8420-failing-loudly-an-empirical-study-of-methods-for-detecting-dataset-shift.pdf.

[47] Roger Ratcliff. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, 97(2):285–308, 1990. https://doi.org/10.1037/0033-295X.97.2.285.

[48] Martin Riedmiller. Neural fitted Q Iteration–First experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328, 2005.

[49] George F. Riley and Thomas R. Henderson. The ns-3 Network Simulator. In Klaus Wehrle, Mesut Güneş, and James Gross, editors, *Modeling and Tools for Network Simulation*, pages 15–34. Springer, Berlin, Heidelberg, 2010. https://doi.org/10.1007/978-3-642-12331-3_2.

[50] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience Replay for Continual Learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 350–360. Curran Associates, Inc., 2019. URL: http://papers.nips.cc/paper/8327-experience-replay-for-continual-learning.pdf.

[51] Noga H. Rotman, Michael Schapira, and Aviv Tamar. Online Safety Assurance for Learning-Augmented Systems. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, HotNets '20, pages 88–95, New York, NY, USA, November 2020. Association for Computing Machinery. https://doi.org/10.1145/3422604.3425940.

[52] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, October 2000. URL: https://www.sciencedirect.com/science/article/pii/S0378375800001154, https://doi.org/10.1016/S0378-3758(00)00115-4.

[53] Ashraf Tahmasbi, Ellango Jothimurugesan, Srikanta Tirthapura, and Phillip B. Gibbons. DriftSurf: Stable-State / Reactive-State Learning under Concept Drift. In *Proceedings of the 38th International Conference on Machine Learning*, pages 10054–10064. PMLR, July 2021. URL: https://proceedings.mlr.press/v139/tahmasbi21a.html.

[54] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. Learning to Route. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, pages 185–191, New York, NY, USA, 2017. ACM. URL: http://doi.acm.org/10.1145/3152434.3152441, https://doi.org/10.1145/3152434.3152441.

[55] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), March 2016. URL: https://ojs.aaai.org/index.php/AAAI/article/view/10295.

[56] Matthias Wichtlhuber, Eric Strehle, Daniel Kopp, Lars Prepens, Stefan Stegmueller, Alina Rubina, Christoph Dietzel, and Oliver Hohlfeld. IXP scrubber: Learning from blackholing traffic for ML-driven DDoS detection at scale. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, pages 707–722, New York, NY, USA, August 2022. Association for Computing Machinery. https://doi.org/10.1145/3544216.3544268.

[57] Gerhard Widmer and Miroslav Kubat. Learning in the Presence of Concept Drift and Hidden Contexts. *Machine Learning*, 23(1):69–101, April 1996. https://doi.org/10.1023/A:1018046501280.

[58] Keith Winstein and Hari Balakrishnan. TCP Ex Machina: Computer-generated Congestion Control. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 123–134, New York, NY, USA, 2013. ACM. URL: http://doi.acm.org/10.1145/2486001.2486020, https://doi.org/10.1145/2486001.2486020.

[59] Sebastien C. Wong, Adam Gatt, Victor Stamatescu, and Mark D. McDonnell. Understanding Data Augmentation for Classification: When to Warp? In *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–6, November 2016. https://doi.org/10.1109/DICTA.2016.7797091.

[60] Zhengxu Xia, Yajie Zhou, Francis Y. Yan, and Junchen Jiang. Genet: Automatic curriculum generation for learning adaptation in networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, pages 397–413, New York, NY, USA, August 2022. Association for Computing Machinery. https://doi.org/10.1145/3544216.3544243.

[61] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. Learning in situ: A randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 495–511, 2020. URL: https://www.usenix.org/conference/nsdi20/presentation/yan.

[62] Francis Y. Yan, Jestin Ma, Greg D. Hill, Deepti Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. Pantheon: The training ground for Internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 731–743, 2018. URL: https://www.usenix.org/conference/atc18/presentation/yan-francis.

[63] Yiding Yu, Taotao Wang, and Soung Chang Liew. Deep-Reinforcement Learning Multiple Access for Heterogeneous Wireless Networks. *IEEE Journal on Selected Areas in Communications*, 37(6):1277–1290, June 2019. https://doi.org/10.1109/JSAC.2019.2904329.

[64] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual Learning Through Synaptic Intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, July 2017. URL: http://proceedings.mlr.press/v70/zenke17a.html.

[65] Qizhen Zhang, Kelvin K. W. Ng, Charles Kazer, Shen Yan, João Sedoc, and Vincent Liu. MimicNet: Fast performance estimates for data center networks with machine learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, pages 287–304, New York, NY, USA, August 2021. Association for Computing Machinery. https://doi.org/10.1145/3452296.3472926.

[66] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, April 2004. https://doi.org/10.1109/TIP.2003.819861.

[67] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 5(5):363–387, 2012. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/sam.11161, https://doi.org/10.1002/sam.11161.

A Puffer: Supplemental Results

Recurring patterns Fig. 9 illustrates that the Puffer traffic does contain patterns that recur at the tail. Each line in this plot corresponds to one batch in the memory assembled by Memento after three weeks of sample selection. The lines show the density of batches with respect to the daily samples over the next 200 days; this captures "how similar the batches in memory are to the traffic of the current day." It shows that some tail batches (the lines with the lowest densities) are sometimes more represented in the daily traffic for a couple of days (see e.g., the step around week 7), then fade away.

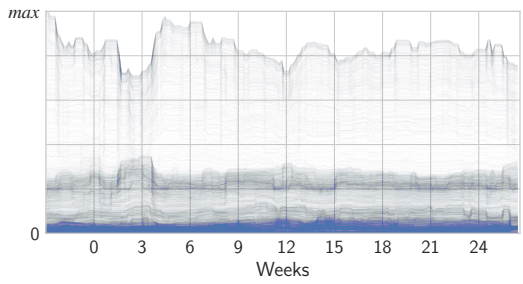


Figure 9: Coverage of batches in the daily samples

Aggregate performance Fig. 14 shows the aggregated SSIM and percent of stream-time spent stalled on a 2D grid in the style of the Puffer [61] publication and website [1]. Concretely, we show two sets of ABRs. First Memento (default and $T = 0$) and Fugu_{Feb}, aggregated since the latest version of Memento was deployed on September 19th 2022 until February 13th, the cutoff for our current evaluation. We cannot aggregate over the deployment duration Fugu as it was discontinued on October 5th. thus, we additionally include a second set consisting of Fugu and Fugu_{Feb}, aggregated from 2020 until Fugu was discontinued.

Timeseries (real-world) Fig. 15 shows QoE results per day over time for Memento with default parameters, Memento ($T = 0$) and for Fugu_{Feb} since September 19th.

Fig. 16 shows the same results, but the mean and bootstrapped 90% CI over a two-week sliding window.

Fig. 17 shows the prediction improvements over Fugu_{Feb} for Memento with default parameters and with $T = 0$.

Past degradation of Memento ($T = 0$) over time In a previous deployment, we observed Memento ($T = 0$) to degrade over time, as shown in Fig. 10. Without forgetting, it kept accumulating noise and retraining. At first, the Puffer control loop around the TTP was able to compensate for this degradation, but as the model became too bad, it failed. Over time, the Image quality degradated of over 30 %.

However, we later discovered an issue in this deployment that prevented Memento from using the deployed models' prediction, effectively disabling BBDR. After fixing this issue, we observed that the performance of Memento ($T = 0$) recovered, highlighting the benefits of considering both prediction and output space. nevertheless, we are again beginning to see the signs of noise accumulation, in particular stronger coverage increase and frequent retraining, and are monitoring the current Memento ($T = 0$) deployment closely.

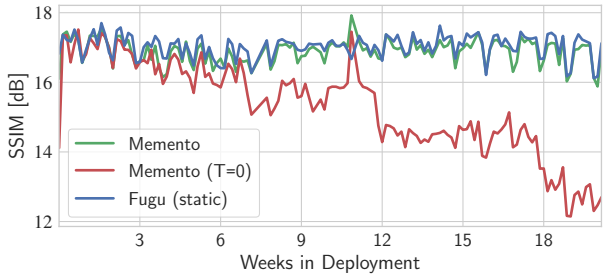


Figure 10: Long-term degradation of Memento ($T = 0$) over time in a previous deployment on Puffer.)

Prediction score improvements compared to the past Fig. 11 shows Fig. 4a overlayed with the prediction score improvements of Fugu compared to Fugu_{Feb} in the past. These curves are not directly comparable, as they come from different periods of time and the underlying data may have shifted. However, we can see that daily retraining with random samples did not consistently improve the TTP tail score; it even worsened the tail score for 20 % of days. This may explain why daily retraining yields significantly smaller tail QoE improvements than retraining with samples selected by Memento, which consistently improves the tail predictions.

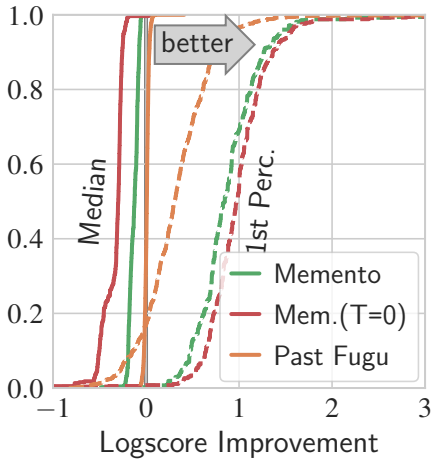


Figure 11: Retraining with daily samples did not consistently improve the tail prediction score in the past.

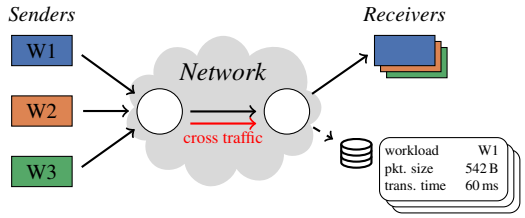


Figure 12: Simulation setup.

Alternative selection metrics Fig. 18 shows further alternative sample selection metrics in addition to loss (§4.4).

Fig. 18a shows the results for model confidence, i.e., the probability of the predicted transmission time bin (Puffer separates the transmit time into 21 bins ranging from 0.125 s to 10 s). With this metric, Memento prefers to discard samples with high confidence to keep 'difficult' samples with low confidence. For high temperatures, performance improvements are small. For low temperatures, we observe strong variation between runs, with more performance degradation than improvement. In summary, this selection metric is unreliable and fails to consistently improve performance.

Fig. 18b shows the results for label counts, a simplified version of density that is often used for classification data. We use the transmission time bin of each sample as the label. With this metric, Memento prefers to discard samples with high label counts to keep 'rare' samples with low label counts. We observe this approach to drastically reduce performance. In the Puffer environment, the majority of samples are assigned to the lowest transmit time bins. Going by label counts alone removes too many of these samples and the model forgets common patterns, similar to the loss metric.

Finally, Fig. 18c shows the results for stalled sessions. With this metric, Memento prefers to discard samples if they belong to a session that did not stall to keep samples from sessions that did stall. We observe consistent improvements, but they are small. It does not provide a fine-grained enough selection to significantly improve tail performance.

Increased memory capacity Fig. 19 shows the results for Memento compared to a random memory with the same capacity (both 1 M) and to a random memory with a double capacity (2 M). The random memory is set up like Fugu, selecting samples randomly over the last two weeks. We can see that simply increasing the capacity fails to address dataset imbalance, and performance is virtually identical at double the training effort.

Timeseries (replay) Fig. 20 shows the evolution of each benchmark experiment over the whole 6-month duration. Fig. 21 and Fig. 22 show the same time series for the experiments with alternative selection metrics, and for combining Memento with MatchMaker and JTT, respectively.

For temperature-related benchmarks, we observe that a high temperature (uniformly random selection) performs slightly worse at the tail than Fugu. This may be because Fugu keeps samples from the past 14 days, while a uniformly random selection phases samples out more quickly.

B Simulation: Supplemental Results

Simulation setup Fig. 12 illustrates the setup we used for the evaluation of Memento in simulation.

Model Architecture We use the following parameters for the classification and regression models used in our simulation experiments. We use supervised training and select the number of layers, neurons, and training parameters via hyperparameter optimization [23].

	Model	
Parameter	Classification	Regression
Hidden Layers	3	4
Hidden Units	512	362
Learning Batchsize	512	512
Learning Rate	5.91×10^{-5}	0.382

Table 1: Model parameters.

We have implemented both models using Keras [12]. For all layers, we use batch normalization [25] and ReLU activation [40]. We train both networks with the Adam optimizer [29] using the default decay parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$. We train for up 200 epochs with early stopping.

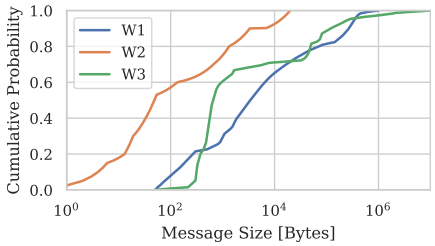
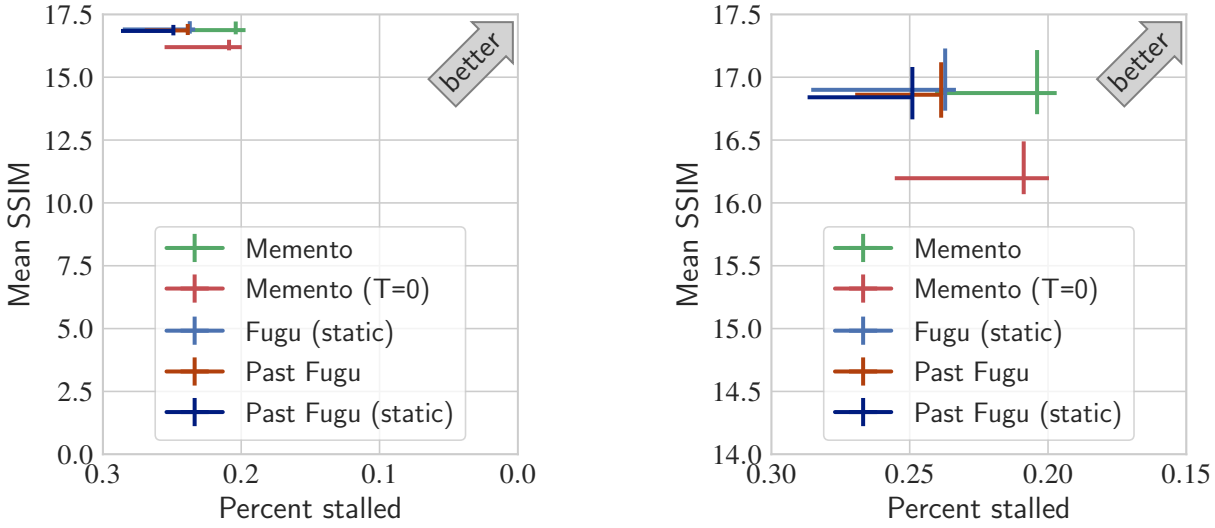


Figure 13: Message size distributions [39].

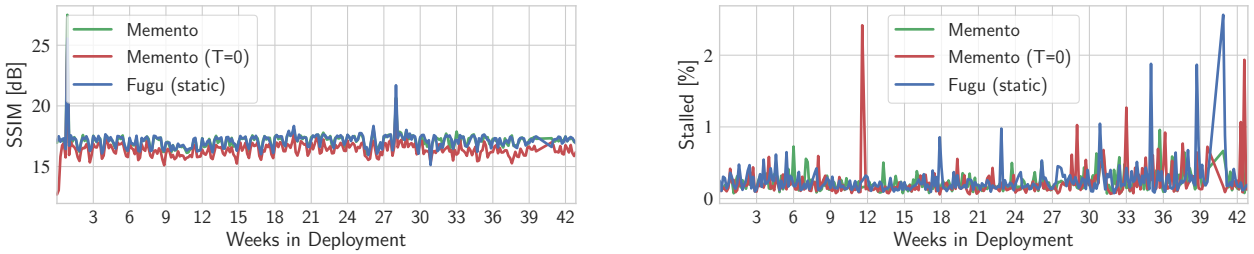
Workload distributions Fig. 13 shows the message size distribution for the workloads we use, published by the HOMA project [39]): Facebook web server (**W1**), DC-TCP (**W2**), and Facebook Hadoop (**W3**). Messages are generated with Poisson-distributed inter-arrival times.



(a) Image quality and Percent of stream-time spent stalled.

(b) Zoomed in.

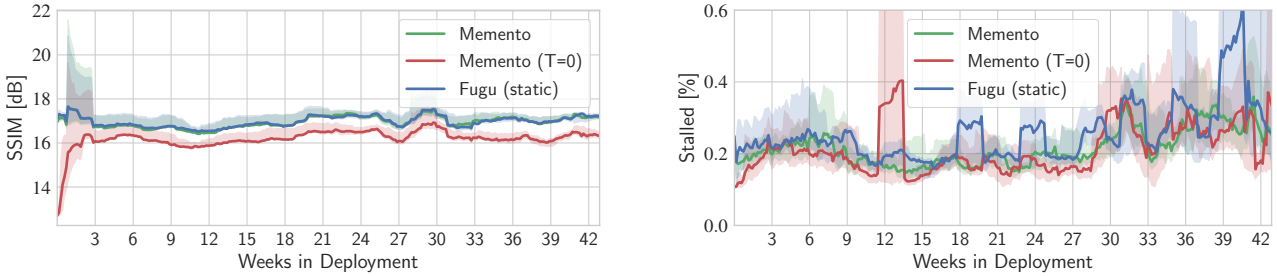
Figure 14: Aggregate performance. Mean and bootstrapped 90% confidence intervals from the deployment of the current version of Memento on September 19th, 2022 until February 13th, 2023. ABRs annotated with 'past' indicate past data from April 9th, 2020 until October 5th, 2022, when Fugu was discontinued.



(a) Image quality, measured in SSIM.

(b) Percent of stream-time spent stalled.

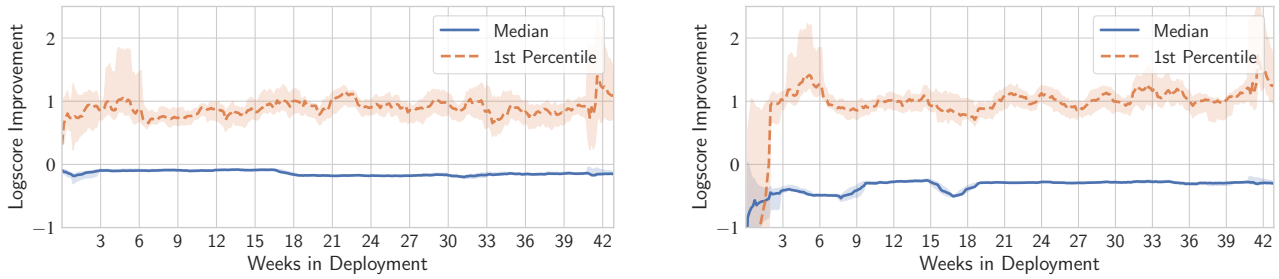
Figure 15: Evolution of QoE metrics over time. Absolute Values for each ABR.



(a) Image quality, measured in SSIM.

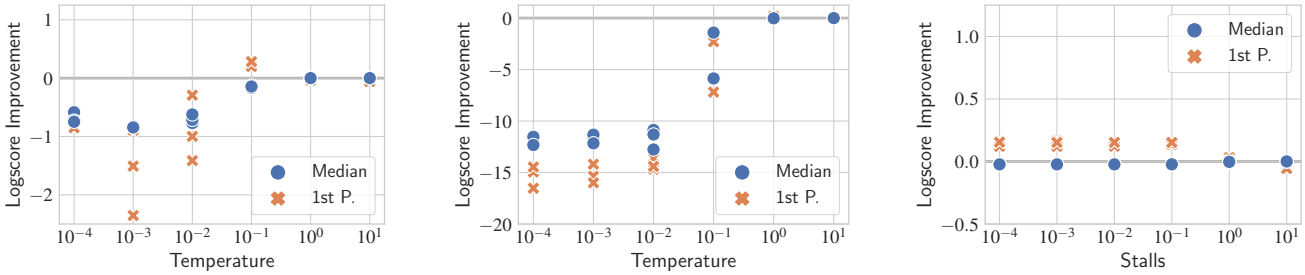
(b) Percent of stream-time spent stalled.

Figure 16: Evolution of QoE metrics over time. Absolute Values for each ABR. Mean and bootstrapped 90% confidence interval over two-week sliding windows.



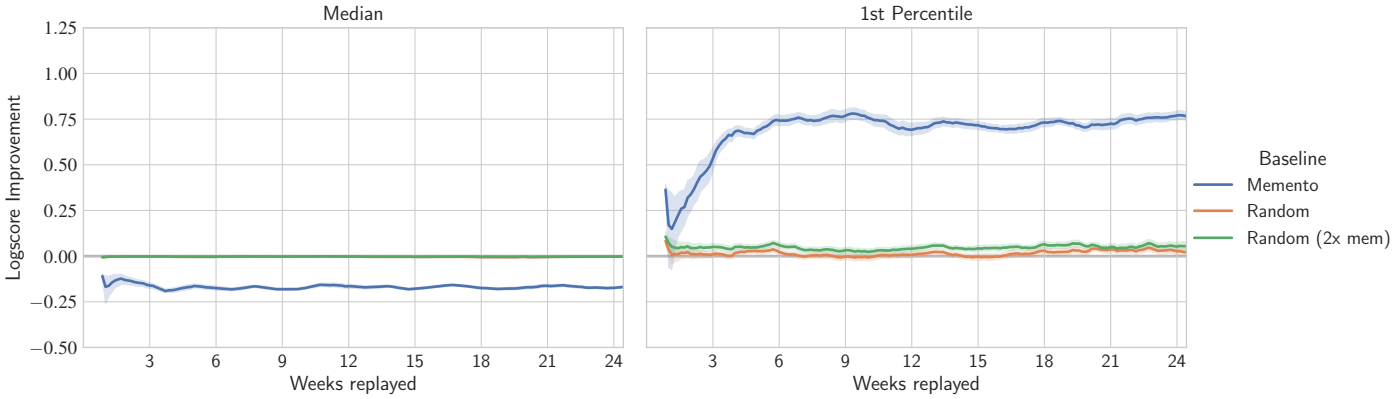
(a) Memento. (b) Memento ($T = 0$).

Figure 17: Logscore improvements compared to Fugu_{Feb}.



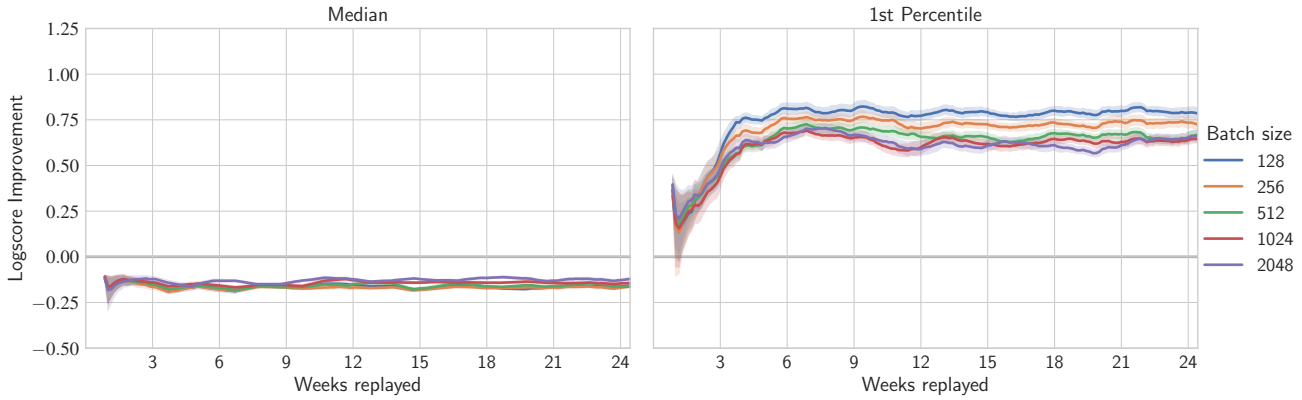
(a) Confidence (b) Label counts (c) Stalled sessions

Figure 18: Additional alternative selection metrics.

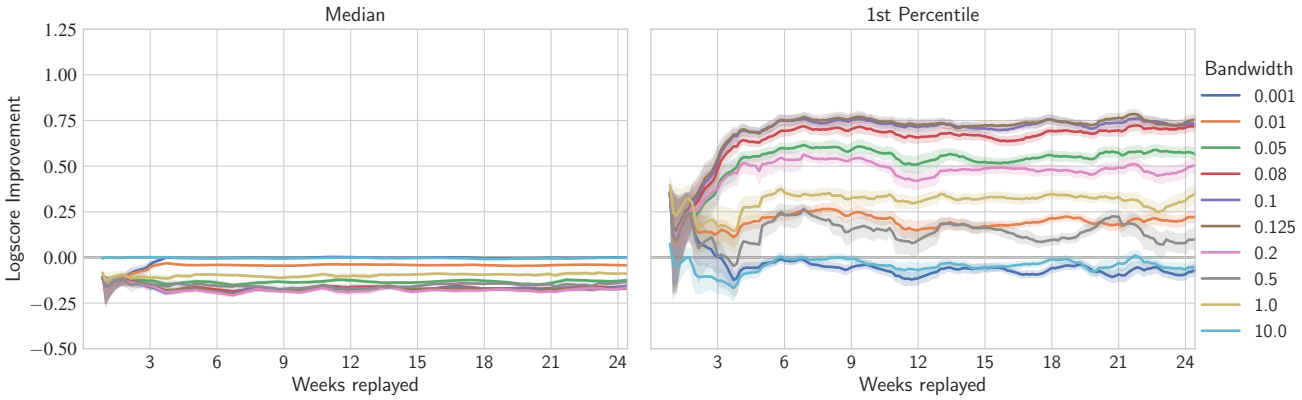


(a) Memento (1 M samples) compared to random sampling like Fugu (1 M and 2 M samples respectively)

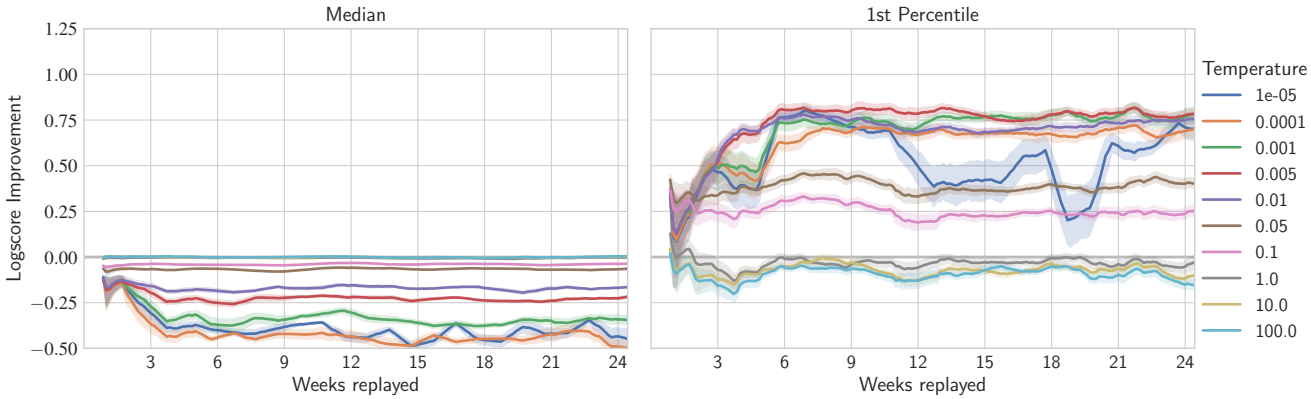
Figure 19: Logscore improvements compared to Fugu, concretely the median and tail (1st percentile) improvements for each day. We compute the mean over all three 6-month replays, and plot again the mean of this value and bootstrapped 90% confidence intervals over two-week windows.



(a) Batch size.

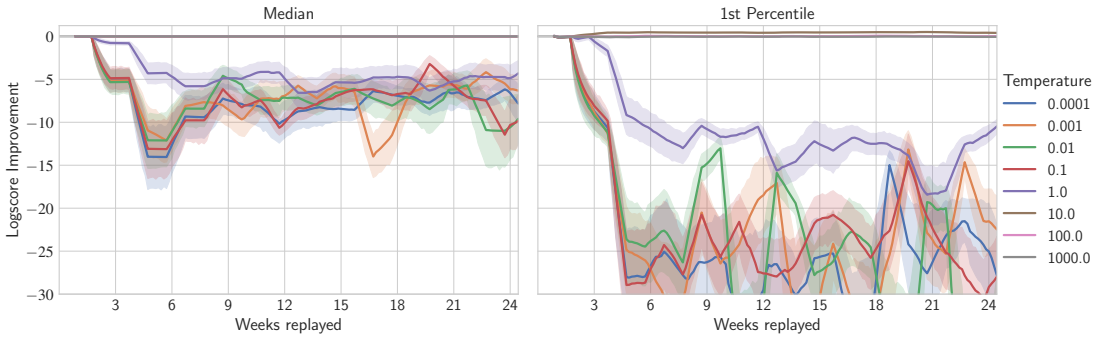


(b) Bandwidth

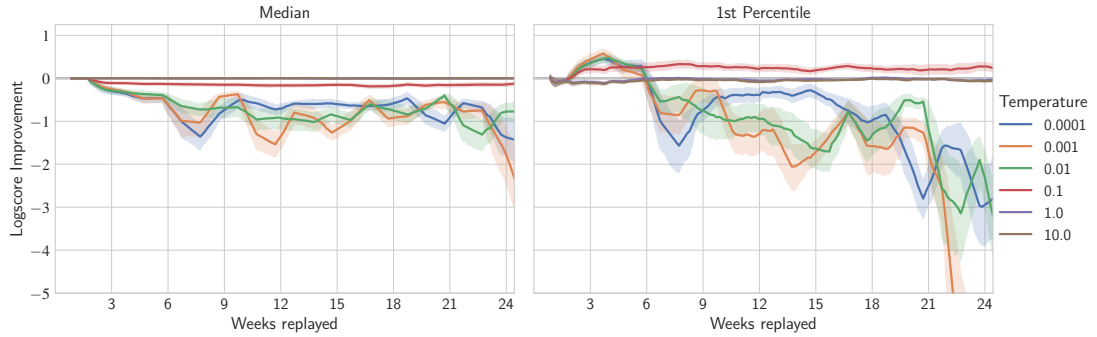


(c) Temperature

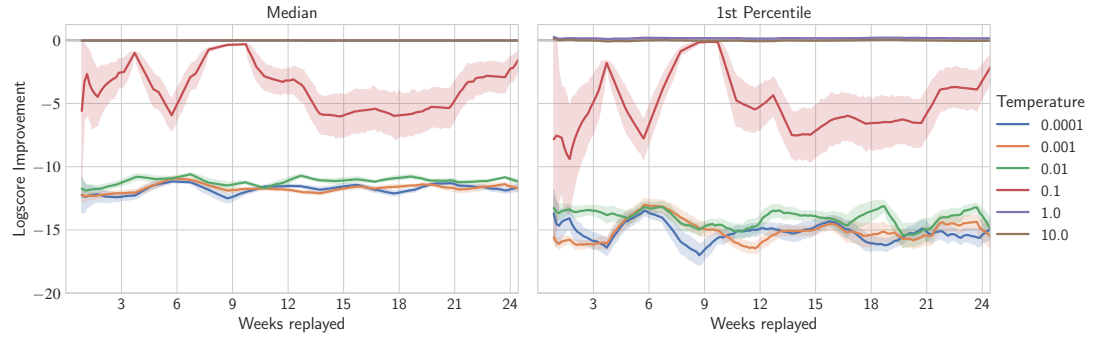
Figure 20: Logscore improvements compared to Fugu (see Fig. 19) for all selection metric experiments.



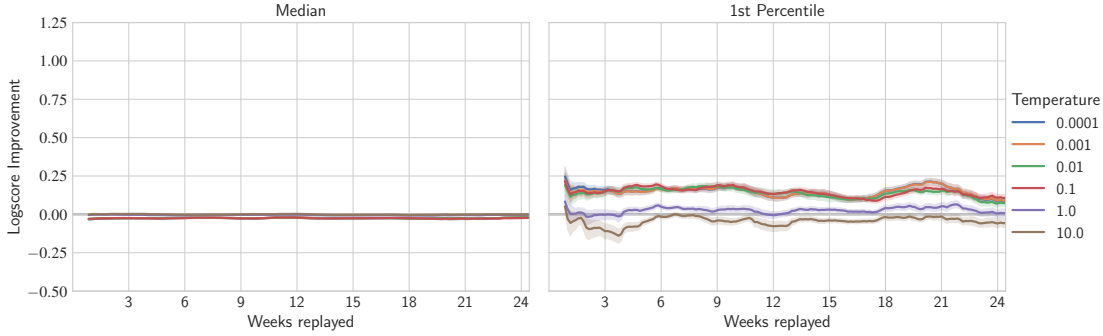
(a) Loss.



(b) Confmednce.

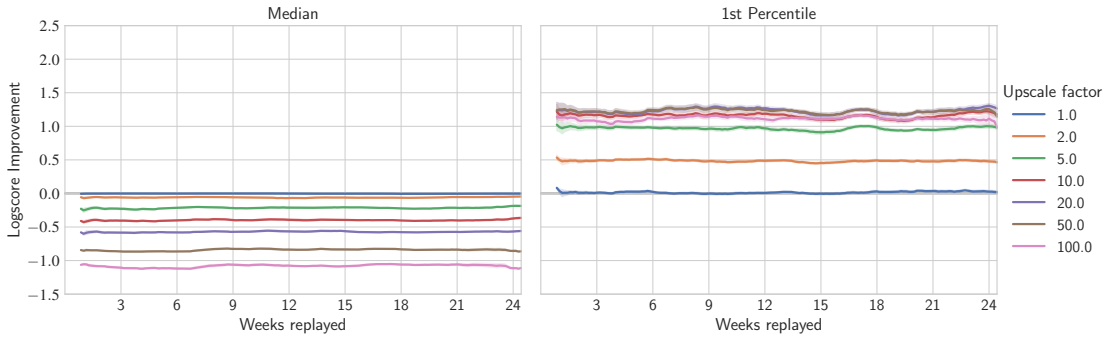


(c) Label counts

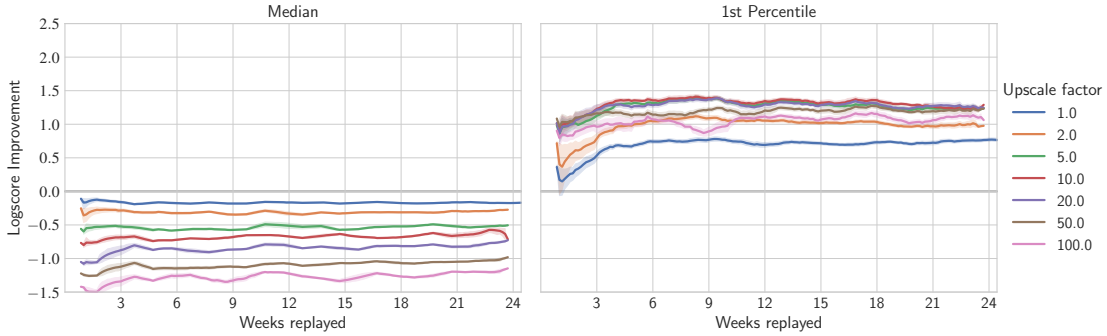


(d) Stalled sessions

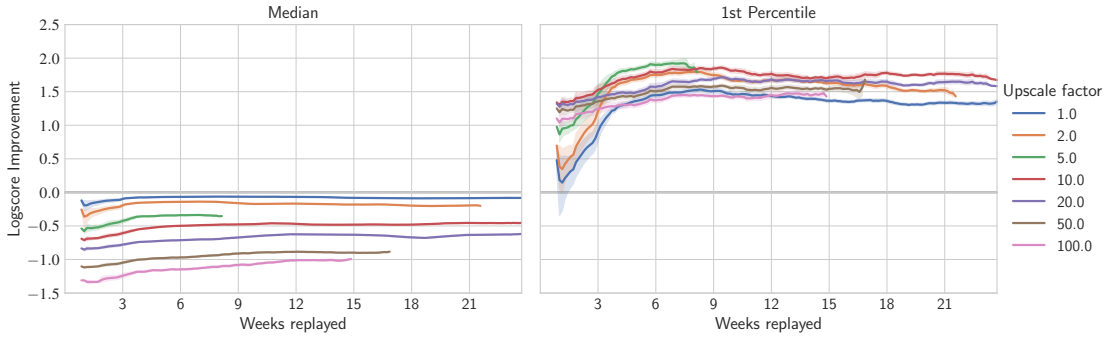
Figure 21: Logscore improvements compared to Fugu (see Fig. 19) for all selection metric experiments.



(a) JTT with random sample selection.



(b) JTT with Memento's sample selection.



(c) JTT with Memento's sample selection and MatchMaker predictions.

Figure 22: Logscore improvements compared to Fugu (see Fig. 19) for all combination experiments.